Mike Etberg

FUNCTIONAL SPECIFICATION
FOR A
DOS 3 DISK CLEANUP UTILITY

Preliminary 20-Feb-81

Prepared by: Harry B. Stewart
NEOTERIC
15816 San Benito Way
Los Gatos, CA 95030
(408) 395-6478

FUNCTIONAL SPECIFICATION
FOR A
DOS 3 DISK CLEANUP UTILITY

1. Introduction

2. Verification processes
   2.1 Non-file processes
       2.1.1 Validation of Volume I.D.
       2.1.2 Validation of Sector Allocation Map
       2.1.3 Validation of Bad Sector Table
       2.1.4 Validation of Boot Sector
       2.1.5 Validation of Directories & entries
       2.1.6 Sector allocation
   2.2 File processes
       2.2.1 FSP #s 1 & 2
       2.2.2 Link SP
       2.2.3 Data portion
       2.2.4 File length
       2.2.5 File allocation
   2.3 Final validation of Sector Allocation Map
   2.4 Medium validation

3. User interface
   3.1 Operating environment
   3.2 Commands and options
   3.3 Messages and responses
       3.3.1 Normal
       3.3.2 Error

4. Limitations

1. Introduction

This document is a preliminary external specification for a DOS 3 type disk maintenance utility. This utility provides the functions of its DOS 2 counterpart, and provides two new functions which automatically verify or correct the structure of a disk.

This version of the specification describes the disk verification process primarily, with the low level functions assumed to be very similar to their DOS 2 counterparts. Later versions of this specification will document the entire external interface. The term 'T.B.D.' indicates an item that is to be determined at a later phase of development.

2. Verification processes

The major effort involved with preparing this utility is the development of the verification process (CHECK command). The remainder of this section discusses the operation of that process.

The verification process can be functionally segmented into several subprocesses, as shown below:

Validation of the Volume I.D.
Validation of the directories
Validation of the individual files in the directories.
Validation of the Bad Sector Table.
Validation of the Sector Allocation Map.

Although most of these subprocesses are independent, the validation of the Sector Allocation Map requires the cooperation of all of the other subprocesses. The technique used is to create a duplicate allocation map, which is initially empty, at the beginning of the process; this map is then updated as sectors are referenced by structural elements. At the end of the entire process, the duplicate map should be identical to the actual Sector Allocation Map, if there are no errors.

2.1 Non-file processes

The first parts of the disk structure to be validated are the general overhead structures that are present on all DOS 3 type disks, and are independent of the actual files present on the disk. These structural elements include:

Volume I.D.
Sector Allocation Map (gross check).
Bad Sector Table.
Boot sector.
Directories.

The subsections that follow discuss the validation techniques that are to be used for each of these structural elements.

2.1.1 Validation of Volume I.D.

Since all other structural elements of the disk are contained in (or pointed to by) the Volume I.D. sector (sector #4), it is important to validate this element first. If there is any chance that sector #4 of the disk being examined is not a valid Volume I.D., then the user is immediately informed and the process is suspended.

The following tests are made in order to validate the Volume I.D.:

VOLTYP (Volume Type byte) is tested for a value of $A5, the only type currently defined; as additional volume types are created, this utility should be modified to handle them. Theoretically, the latest version of the utility should be able to handle all DOS 3 volume types. -Table

VOLLAB (Volume Label) is written to the screen and is tested to see that it contains only ATASCII charcters in the range of $20 through $7F.

VOLDES (Volume Description) is written to the screen and is tested to see that it contains only ATASCII characters in the range of $20 through $7F.

The numeric values of VOLMAX (Maximum Sector Number), VOLNAL (Number of sectors in allocation unit), VOLFLG (Volume Flags), VOLTRY (Volume Retry Count) and VOLOSN (O.S. Revision #) are written to the screen for operator verification. — FULL

VOLTNS (Total Number of Sectors) and VOLNAB (VOLTNS/8) are checked to see that they are correctly derived from the value of VOLMAX. Where VOLTNS = ~~(VOLMAX + 7) MOD 8, and VOLNAB = VOLTNS / 8.~~ (((VOLMAX+VOLNAL-1)/VOLNAL)+7)/8*8 & VOLNAB = VOLBMO + VOLTNS/8.

VOLBM (Starting Sector of Sector Allocation Map) is checked to see that the sector number is not greater than the value of VOLMAX.

VOLBMO (Sector Allocation Map Offset) is checked to see that the Sector Allocation Map does not improperly overlay another structural element.

VOLBST (Starting Sector of Bad Sector Table) is checked to see that the sector number is not greater than the value of VOLMAX.

VOLBSO (Bad Sector Table Offset) is checked to see that the Bad Sector Table does not improperly overlay another structural element.

VOLOSS (O.S. Start Sector) is checked to see that it does not exceed the value of VOLMAX.

VOLNOS (O.S. File Length) is checked to see that ... (T.B.D.).

VOLNDR (Number of Directories) is checked to see that its value is between 1 and 16.

Further checking of Volume I.D. elements is discussed in the remaining subsections of 2.1.

2.1.2 Validation of Sector Allocation Map

A preliminary validation of the Sector Allocation Map is made to assure that allocations exist for the following structural elements (most of whose locations are defined in the Volume I.D.):

Volume I.D. sector itself (sector #4).
Sector Allocation Map.
Bad Sector Table.
Directories (1-16).
Boot sector (sector #1).

2.1.3 Validation of Bad Sector Table

The Bad Sector Table consists of pairs of sector numbers, all of which must not exceed the value of VOLMAX. The maximum number of entries in the map is contained in VOLNBS.

2.1.4 Validation of Boot Sector

The first six bytes of the Boot Sector will be validity checked as follows:

BOOT FLAG (Byte Ø) must be equal to $Ø1. If this is not the case, no further tests are made.

BOOT LOAD ADDRESS (Bytes 2-3) is checked for ... T.B.D.

INIT ADDRESS (Bytes 4-5) is checked for ... T.B.D.

2.1.5 Validation of Directories & entries

Each of the up to 16 Directory Descriptions are checked as described below:

VDIFLG (Directory Flags) will be either $ØØ for an inactive directory or $Ø1 for an active directory. The rest of the tests described in this subsection apply only to active directories.

VDILBL (Directory Label) is written to the screen and tested to see that it contains only ATASCII characters in the range of $2Ø through $7F.

VDISS (Directory Starting Sector) is checked to see that it does not exceed the value of VOLMAX.

VDINS (Number of Sectors in Directory) is checked to see that its value is between 1 and xx (T.B.D.).

Each of the up to 16 directories defined in the Volume I.D. is read and checked; the following items are examined:

Flags byte.
File name/extension.
File length.
Segment pointers.

FILFLG (File Flags) is checked to see if a file exists for that entry; if not, the remaining tests are skipped. If the file exists but has not been properly CLOSED (OPO = 1), then ...
T.B.D. should check IOCB to see if open for that file, else prompt to close

FILNAM/FILEXT (File Name/Extension) are checked to see that the names consist only of ATASCII characters ranging from $20 through $7F.

FILLEN (File length) is checked to see that the file length specification does not exceed the size of the disk; the value of the length / 256 must not exceed the value of VOLMAX. The type must be Length. how about must not exceed # of allocated sectors? in VOLMAX

FILSP1-2 (File Segment Pointers 1-2) are checked for validity by examining the 3 fields of each pointer:

SPTYPE (Segment Pointer Type) must be one of the 3 valid types for a data segment pointer: Null, Block or EOF.

SPSEC (Segment Starting Sector) must not exceed the value of VOLMAX.

SPCNT (Segment Pointer Count) -- SPSEC + SPCNT must not exceed the value of ~~VOLMAX~~. # of allocated sectors

FILSP3 (LINK SP) is checked for validity by examining the 3 fields of the pointer:

SPTYPE (Segment Pointer Type) must be one of the 2 valid types for a link segment pointer: Null or Link. If the type is Null, then the remaining tests are skipped.

SPSEC (Segment Starting Sector) must not exceed the value of ~~VOLMAX~~. # of allocated sectors

SPCNT (Segment Offset) must be equal to 4.

2.1.6 Sector allocation

As a structural element is examined , a check is made to see that the individual sectors which represent the element are all allocated in the Sector Allocation Map. In addition a check is made to see that the sectors are not yet allocated in the duplicate map; if they are, it indicates that two or more structural elements have sectors in common, a situation that is usually invalid. Note that some elements may have common allocation (the Sector Allocation Map and the Bad Sector Table may each overlap other structures) and will have to be treated specially.

2.2 File processes

Once the general overhead structures of the disk are validated, then the individual directory entries and the associated files may be validated. During this process the duplicate Sector Allocation Map is maintained; at the end of the file verification process the duplicate map should be identical to

the real map. The following tests are included in the file validation process:

All sector pointers valid.
File length valid.
All data and overhead sectors allocated properly.

2.2.1 FSP #s 1 & 2

The Segment Pointers which were validated earlier per section 2.1.5 are now used to locate the data portion of the file. The following conditions are tested for and reported if present:

A Null type Segment Pointer found prior to an EOF type Segment Pointer. This condition probably indicates a file that was not closed properly.

A Length or Link type Segment Pointer found.

2.2.2 Link SP

If a file utilizes more than two segments then the Link SP (FILSP3) of the directory entry for that file will point to a Segment Pointer Sector which will contain more segment definitions and may in turn point to another Segment Pointer Sector. This segment pointers within this structure must be analyzed in the same manner as FSP #s 1 and 2. The following conditions are tested for and reported if present:

A Link SP is defined and either FSP 1 or 2 was an EOF type Segment Pointer.

Neither FSP 1 nor 2 were an EOF type Segment Pointer and the Link SP is a Null type.

The type of FILSP3 is other than Null or Link .

The backward Link Segment Pointer in a Segement Pointer Sector does not link back to its predecessor sector.

2.2.3 Data portion

Because there is absolutely no overhead or redundancy in the data portion of a file, there is no way to determine whether a given data portion allocation belongs to a given file. This is the single biggest deficiency in the DOS 3 files structure with regards to integrity check and correction.

2.2.4 File length

The file length as specified in FILLEN (File Length) will be checked to see that is consistent with the file length as derived from the Block and EOF type Segment Pointers.

2.2.5 File allocation

As a file is traversed, a check is made to see that the individual sectors which represent the file are all allocated in the Sector Allocation Map. In addition a check is made to see that the sectors are not yet allocated in the duplicate map; if they are, it indicates that two or more files have sectors in common, a situation that is invalid.

2.3 Final validation of Sector Allocation Map

Once the non-file and file verification processes are complete, the duplicate Sector Allocation Map should be identical with the real Sector Allocation Map. Discrepancies would be due to one or more of the following conditions:

Unused allocation -- An allocation is contained in the map which is not utilized by any structural element or file.

Unallocated usage -- A structural element or file utilizes a disk area that has not been allocated. This will have been reported earlier, during the non-file and/or file validation processes.

2.4 Medium validation

The portions of the medium utilized within existing structures are obviously validated in the earlier processes; however, as a final test, each sector on the disk is read and all read errors are catalogued. The Bad Sector Table is compared with the result of this test and any discrepancies are reported.

3. User interface

3.1 Operating environment

It is anticipated that the early versions will be menu driven and will be similar in operation to the DOS 2 file fix program. An option to have the program be command driven as well as menu driven will be added later.

3.2 Commands and options

The following commands are available:

CHECK -- Runs all of the tests described in section 2, but makes no attempts to correct any of the problems found.

FIX -- Runs all of the tests described in section 2, and attempts to correct problems as they are found.

DIRECTORY LISTING -- Displays all, or portions of, one or more of the directories on the disk.

TRACE FILE ALLOCATION -- Displays all of the sector numbers allocated for a specified file; the information is obtained from the file Segment Pointers.

MODIFY DIRECTORY ENTRY -- Allows the operator to modify portions of a directory entry.

SET DRIVE NUMBER -- Allows the operator to set the default disk drive number.

RETURN TO DOS

DISPLAY SECTOR (HEX, DECIMAL OR ATASCII)

PATCH SECTOR (HEX, DECIMAL OR ATASCII)

~~SET OUTPUT DEVICE (IE. P:, D:, E:)~~ okay ---- below

The following options are available:

VERIFY ON/OFF -- When FIX is running, this option determines whether operator verification is required for trivial fixes. good! Operator verification is always required for non-trivial fixes. The default value is ON.

REPORT FULL/BRIEF/OFF -- When CHECK or FIX is running, this option determines whether a progress report is generated, and if generated, whether it is to be a full report or a brief report. Errors are reported independently of this option. The default value is BRIEF.

FILE TRACE ON/OFF -- When CHECK or FIX is running, this option determines whether an annotated trace of all disk sector references is generated. The default if OFF.

SELECT REPORT DEVICE -- The default report device is the Screen Editor, but the report may be optionally output to any device other than the disk being tested.

NUMERIC RADIX -- The default number radix for all numbers read or written is hexadecimal, but the operator may change the radix to any value from 2 to 16. The radix specification itself is always in decimal notation. Thus binary is 2, octal is 8, decimal is 10, hexadecimal is 16, etc. Note that the reports are designed for hexadecimal reporting to a 38 character screen; hence some columnar data may not line up properly when using other notations, due to screen wrapping.

BAD SECTOR SCAN ON/OFF -- The default case is to perform a bad sector scan as part of a CHECK or FIX command. This operation may optionally be eliminated.

who owns sector X -- -

3.3 Messages and responses

This section lists most of the anticipated messages that will be generated by the program; the messages that are being brought forward from the DOS 2 disk fixer are not shown here at this writing (they will appear in later versions of this document).

3.3.1 Normal

One class of messages is generated as a reporting function to allow the user to follow the progress of a given command. These messages are summarized below, along with an indication as to whether they are part of the BRIEF or FULL report:

```
VOLUME 'llllllll' TYPE tt SECTORS sss          BRIEF/FULL/OFF
```

This message is written at the beginning of any command.

Where: l is the volume label (VOLLAB).
t is the volume type (VOLTYP).
s is the number of sectors on the disk (VOLMAX).

```
DESCR 'ddddddddddddddddddddddddddddddd'        FULL
```

This message is written at the beginning of any command.

Where: d is the volume description (VOLDES).

```
SECTOR ALLOCATION MAP AT aaa/oo                FULL
BAD SECTOR TABLE AT      bbb/oo nn
DIRECTORY 1 AT           ddd    nn
DIRECTORY 2 AT           ddd    nn
etc.
```

This message is written during the Volume I.D. validation phase.

Where: a is the starting sector for the Sector Allocation Map.
o is the sector offset.
n is the number of entries (max) for the element.
b is the starting sector for the Bad Sector Table.
d is the starting sector for a Directory.

DIRECTORY nn '11111111' BRIEF

This message is written prior to the examination of each disk directory entry.

Where: n is the directory number (1-16 by position).
l is the directory label (VDILBL).

DIRECTORY nn '11111111' FULL
FLAG ff START sss SECTORS qqq

This message is written prior to the examination of each disk directory entry.

Where: n is the directory number (1-16 by position).
l is the directory label (VDILBL).
f is the flag byte value (VDIFLG).
s is the starting sector for the directory (VDISS).
q is the number of sectors in the directory (VDINS).

FILE 'nnnnnnnnnnnn.eee' FLAG ff FULL

This message is written prior to the examination of each file within a directory.

Where: n is the file name (FILNAM).
e is the file extension (FILEXT).
f is the file flag byte (FILFLG).

FSP 1 TYPE tt SECTOR sss COUNT nnn TRACE
FSP 2 TYPE tt SECTOR sss COUNT nnn
LINK SP tt SECTOR sss OFFSET oo
SP TYPE tt SECTOR sss COUNT nnn
etc.

This message is written during the validation of the file structure if the FILE TRACE option is ON (CHECK or FIX), or during the execution of the TRACE FILE ALLOCATION command.

Where: t is the Segment Pointer type.
s is the sector number.
n is the sector count.
o is the link offset.

BOOT SECTOR: FLAG ff SECTORS nn FULL
LOAD ADDR llll INIT ADDR iiii

This message is written at the time the Boot Sector is validated.

Where: f is the flag byte.
n is the number of consecutive boot sectors.
l is the load address.
i is the initialization address.

FREE SECTORS sss BRIEF/FULL/OFF

This message is written at the end of a CHECK or FIX operation.

Where: s is the number of unallocated sectors on the disk.

BAD SECTORS ssss/mmmm ssss/mmmm ... BRIEF/FULL/OFF

This message is written at the end of a CHECK or FIX operation; the information is obtained from the Bad Sector Table.

Where: s is the number of a known bad sector.
m is the corresponding good sector to which it is mapped.

3.3.2 Error

Another class of messages is generated only upon detection of an anomoly in the disk structure. These messages are summarized below:

CAN'T PROCESS VOLUME TYPE tt

This message is generated if the volume type is not one that can be processed by the CHECK or FIX commands; the process will suspend itself.

Where: t is the volume type (VOLTYP).

INVALID SECTOR POINTER FOR vvvvvv SECTOR sss

This message is written whenever a sector pointer data type has an out of range value (greater than the value of VOLMAX).

Where: v is the symbolic name for the variable in question.
s is the value of the sector number in question.

VARIABLE vvvvvv OUT OF RANGE: IS iii S/B rr xxx

This message is written whenever a variable has an out of range value.

Where: v is the symbolic name for the variable in question.
i is the value in question.
r is a relation operator (< > = etc.).
x is the expected value or limit.

TEXT CONTAINS INVALID CHARACTERS

This message is written whenever a text field is encountered which contains invalid ATASCII characters (not in the range of $20 through $7F).

PRIOR ALLOCATION FOR SECTOR sss

This message is written whenever a structural or file element is encountered which utilizes a sector which has already been allocated by another element.

Where: s is the sector in question.

NO ALLOCATION FOR SECTOR sss

This message is written whenever a structural or file element is ecountered which is not currently allocated.

Where: s is the sector in question.

MISMATCH BETWEEN ACTUAL ALLOCATION
MAP AND COMPUTED ALLOCATION MAP,
REPLACE WITH COMPUTED MAP (Y/N)?

This message is generated at the end of the FIX process whenever the duplicate allocation map is not identical with the Sector Allocation Map.

READ ERROR ee SECTOR sss

This message is generated whenever a disk read error is encountered. The program will attempt to reread up to 3 times before giving up.

Where: e is the error status returned by CIO.
s is the sector number.

LINK SP MISMATCH ppp sss

This message is generated whenever a mismatch is found between the backward Link SP of a Segment Pointer Sector (s) and the forward Link SP of its predecessor (p).

4. Limitations

The primary limitation in the current design is the inablity to determine the true ownership of a file data sector where there are multiple claims via Block or EOF Segment Pointers. The problem might be resolved by adding some form of redundant information to the FMS disk structure, but the nature of such information has not yet been identified. The DOS 1 structure was well covered in this area, having both a file relative record number and a file identifier within every data sector.